The Journal of Systems and Software xxx (2009) xxx–xxx

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

SALSA: QoS-aware load balancing for autonomous service brokering

Bas Boone [a,*], Sofie Van Hoecke [a], Gregory Van Seghbroeck [a], Niels Jonckheere [b], Viviane Jonckers [b], Filip De Turck [a], Chris Develder [a], Bart Dhoedt [a]

[a] INTEC Broadband Communication Networks (IBCN), IBBT, Ghent University, Belgium
[b] System and Software Engineering Lab (SSEL), Vrije Universiteit Brussel, Belgium

A R T I C L E I N F O

Article history:
Received 24 December 2008
Received in revised form 22 June 2009
Accepted 15 September 2009
Available online xxxx

Keywords:
Load balancing
Weighted round-robin
Autonomous system
Service brokering
Simulated annealing
High throughput

A B S T R A C T

The evolution towards "Software as a Service", facilitated by various web service technologies, has led to applications composed of a number of service building blocks. These applications are dynamically composed by web service brokers, but rely critically on proper functioning of each of the composing subparts which is not entirely under control of the applications themselves. The problem at hand for the provider of the service is to guarantee non-functional requirements such as service access and performance to each customer. To this end, the service provider typically divides the load of incoming service requests across the available server infrastructure. In this paper we describe an adaptive load balancing strategy called SALSA (Simulated Annealing Load Spreading Algorithm), which is able to guarantee for different customer priorities, such as default and premium customers, that the services are handled in a given time and this without the need to adapt the servers executing the service logic themselves. It will be shown that by using SALSA, web service brokers are able to autonomously meet SLAs, without a priori over-dimensioning resources. This will be done by taking into account a real time view of the requests by measuring the Poisson arrival rates at that moment and selectively drop some requests from default customers. This way the web servers' load is reduced in order to guarantee the service time for premium customers and provide best effort to default customers. We compared the results of SALSA with weighted round-robin (WRR), nowadays the most used load balancing strategy, and it was shown that the SALSA algorithm requires slightly more processing than WRR but is able to offer guarantees – contrary to WRR – by dynamically adapting its load balancing strategy.

1. Introduction

Nowadays, many newly conceived applications are constructed through integration of already available service components. The approach is made possible through the Service Oriented Architecture (SOA) and "Software as a Service" paradigm, typically using web service technologies to publish, discover and integrate service components. This technology also allows to replicate web services on new servers to scale in response to the needed demands. SOA structures large applications as collections of web services from inside and outside the company, resulting in greater flexibility and uniformity. As a result customers no longer buy software for permanent in-house installation but only buy services as needed. Since an increasing number of third-party software companies are offering web services on a commercial basis, SOA systems may consist of such third-party services combined with others created in-house.

Instead of hard coding service calls in the customer's source code, brokers provide dynamic service selection to automatically select and seamlessly link the services in order to meet the business system requirement, optimize response times or reduce the costs. By using web service brokers, customers only have to interact with the service broker, hiding the complexity of selecting the appropriate service. These web service brokers keep the services available for every user and fulfill their requests as quickly as possible.

In a commercial application typically a Service Level Agreement (SLAs) can be mediated between the customers and the service providers defining the functional and non-functional requirements such as the levels of availability, performance, billing and even penalties in case of violation of the SLA. Often, a service provider also wants to service a class of customers on a best effort basis. In the case of performance, the SLA usually specifies constraints on the response time. If no special precautions are taken, unexpected request patterns can drive a web server into overload, leading to poor performance since the server is unable to keep up with the demands, resulting in increased response times. Service providers can solve this problem by over-dimensioning their resources and provide dedicated servers for premium customers to meet

* Corresponding author. Tel.: +32 9 33 14 979; fax: +32 9 33 14 899.
E-mail addresses: bas.boone@intec.ugent.be (B. Boone), sofie.vanhoecke@intec.ugent.be (S. Van Hoecke), gregory.vanseghbroeck@intec.ugent.be (G. Van Seghbroeck).

0164-1212/$ - see front matter © 2009 Published by Elsevier Inc.
doi:10.1016/j.jss.2009.09.033

their SLAs. Due to the diversity, size and non-intrusiveness of service-oriented architectures, stress-test evaluations are not possible to predict behavior under load, leaving the brokering somewhat speculative. Consequently, without dedicated servers for premium customers, intelligent autonomous service brokering is needed in order not to penalize the premium customers of the services and guarantee their SLAs, while at the same time providing best effort to the default customers.

Within this paper, two requirements for service brokers will be fulfilled: on one hand, the broker should be able to autonomously guarantee constraints on the response time by fulfilling a n-percentile on the response time, i.e. the value for which at most n% of the response times are fulfilled in less than that value. On the other hand, brokering should be transparent for the actual servers executing the service. The latter makes sure that the load balancing logic needs only to be implemented in the broker, and standard server software can be used on the servers. This way, no requirements have to be imposed to the (external) service providers within distributed service-oriented architectures.

In order to fulfill these requirements, the Simulated Annealing Load Spreading Algorithm (SALSA) presented in this paper can load balance requests, and selectively drop some requests from the default users to reduce the web servers' load in order to guarantee SLA to premium customers and provide best effort to default customers (see Fig. 1). SALSA provides QoS-aware load balancing for autonomous service brokering since the SLAs are only mediated between the customers and the QoS-aware broker. As a result, customers do not have to mediate SLAs with the increasing number of service providers and service providers can be QoS unaware and are released from mediating SLAs.

The presented algorithm can be applied in a wide range of application areas. For example multimedia content delivery can benefit from autonomous service brokering in order to meet premium guarantees (for e.g. subscribed customers). The service broker can dynamically select the needed services (e.g. services for broadcasting, streaming, payment and security) in order to set up a video-on-demand stream meeting the request (e.g. high quality, no delay or limited output device) of subscribed customers while the non-subscribed customers will have a best effort stream. Another case can be found in eCommerce, where a call center for example negotiates with multiple credit checkers, in order to acquire payment validation. Based on the call center load, the service broker can divide the requests over multiple credit checkers in order not to lose or displease premium clients. Ehealth, where multiple care providers are integrated, is another case that can benefit from SALSA service selection since emergency services and alarm processing services should receive higher priority and guaranteed execution times.

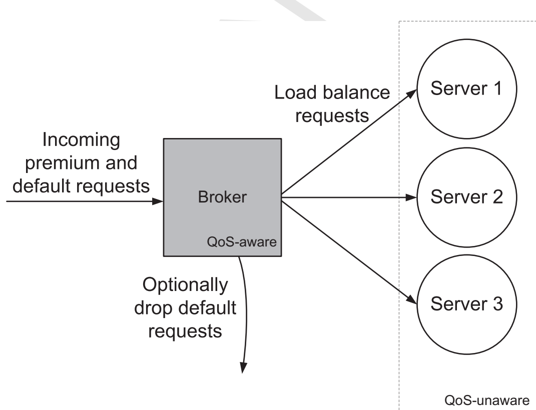


Fig. 1. Objective of the simulated annealing load spreading algorithm.

The remainder of this paper is structured as follows: Section 2 describes the related work, while in Section 3 the theoretical discussion and a criterion to check for optimality is presented. Section 4 describes the SALSA algorithm in more detail. The evaluation results are presented in Section 5. Finally, in Section 6, we will highlight the main conclusions and identify future work.

2. Related work

Web service brokers dynamically select services to fulfill requests based on the user's QoS requirements. In Nahrstedt and Smith (1995) a QoS broker model is described for general distributed systems. However, this broker does not support flexible service selection. In Tao and Lin (2004), a Web service architecture supporting QoS is presented. However, once the services are selected and the link is established, the client communicates with the server directly without any broker intervention during the actual service process. In Verheecke (2007), the Web Service Management Layer (WSML) is presented using Aspect-Oriented Programming (AOP) as a mediation layer between the client and the services. Amongst other, these broker platforms select services based on known quality criteria such as average latency time, execution cost or repudiation (Garofalakis et al., 2006), using Multiple Choice Knapsack (MCK) (Tao and Lin, 2004) or m-dimensional QoS vectors (Liu et al., 2004). When QoS parameters, such as response time, can not be guaranteed by the service providers themselves, the current solutions can not be used to dynamically select the correct service in order to guarantee QoS constraints since neither of these broker solutions is able to adapt to the dynamic server load.

In these cases, web service brokers typically use load balancing (Grosu et al., 2002; Zhang et al., 2001; Cortes et al., 1999) to improve web servers' performance (Bryhni et al., 2000; Cardellini et al., 1999). In Shirazi et al. (1995) a survey of load balancing algorithms is presented. Currently, round-robin is the most used load balancing solution, alternating in a deterministic way between the different service endpoints. This algorithm is successfully applied in DNS servers, peer-to-peer networks, and many other multiple-node clusters/networks. Since all servers are treated equally, all the service endpoints will be invoked an equal number of times, regardless of the response times of the servers. Round-robin is especially suited for brokering when the different service endpoints have (almost) the same response times. If the service endpoints have different response times, weighted round-robin can be used to compensate for these differences. There, servers are presented client requests in proportion to their weighting resulting in fairly distributing the requests amongst service endpoints, instead of equally distributing the requests.

More successful and accurate load balancing requires the web service broker to have some notion of the server load (Di Stefano et al., 1999) in order to adapt the load balancing weight to the current load. This can be done by either time based polling the servers or monitoring their behavior. A round-trip load balancing algorithm monitors the time elapsed between request to the server and response to the client. The average elapsed time of all requests during a sliding window is calculated and the server with lowest calculated average load is selected.

When most requests on the web service broker are of the same kind, round-trip time based load balancing algorithms will not outperform (weighted) round-robin. If however the round-trip algorithm can accurately predict the current load on the servers, this algorithm will be able to distribute the load better when requests are heterogeneous and handle high-load conditions. Both (weighted) round-robin, and round-trip load balancing provide best effort and can not handle priorities, nor guarantee SLAs. Current solutions for priority based load balancing consists of two types of queues, one for default requests and one for premium requests. Requests from the default queue are only handled when there are no premium requests queued or when a certain time is passed (preventing the default requests from starvation if there are always premium requests queued). A combination of priority queueing and weighted round-robin is priority-weighted round-robin presented in Zhang and Harrison (2007). These priority queuing load balancing strategies however do not ensure total response time guarantees to premium customers.

For premium customers best effort is however not good enough. There is a need for service brokers taking into account QoS in order to ensure total response time and prioritize premium customers. In order to meet SLAs for premium customers, dedicated servers can be used. Over-dimensioning the resources can enable a high QoS, but is an expensive option and leads to a waste of capacity.

Web service brokers must support adaptivity to implement autonomous load balancing (Nami and Bertels, 2007; Kephart and Chess, 2003; Sterritta et al., 2005) in order to handle dynamic request loads without a priori over-dimensioning the service provider's resources. Therefore in this paper, we study how autonomous load distribution can adapt to unexpected traffic and sudden load peaks, and compare the results with weighted round-robin.

3. Theoretical model

In this section, the theoretical background and objective are given for the SALSA load balancing algorithm, which is described in more detail in Section 4. We consider the system as depicted in Fig. 2. The broker acts as a statistical switch that randomly forwards client requests to a server with a given probability (similar to WRR); the SALSA algorithm dynamically updates these probabilities to adapt to changing loads. The broker can also selectively drop some requests from the default users to reduce the servers' load in order to guarantee the SLA to premium customers and provide best effort to default customers.

3.1. Problem statement

According to Ardagna et al. (2008), Kanodia and Knightly (2000) and Levy et al. (2003), the web servers are modeled as M/M/1 queueing systems (Gross and Harris, 1998) to compute response times of the Web service requests. A Poisson arrival process is assumed. As illustrated in Christodoulopoulos et al. (2007), the Poisson process is a very good approximation for the arrival process of

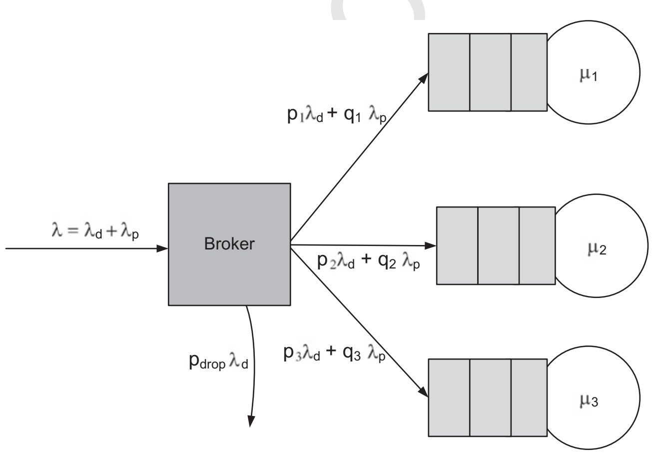


Fig. 2. The SALSA theoretical model.

service requests within a distributed broker platform where the number of service requests is very large, a single requests requires only a very small percentage of the provider's resources and all requests are independent. In Roberts (2001) it is also argued that, while IP packet arrivals can not be accurately modeled as a Poisson process, the arrival of flows on the Internet can generally be approximated as a Poisson process.

The broker is modeled as a statistical switch that randomly forwards client requests to a server with a given probability. This ensures that, if the arrival process towards the broker is a Poisson process, the arrival processes to the web services are also Poisson processes.

The inputs of the problem are defined as follows:

- k: number of web services,
- μ_i: processing intensity for web service i. This parameter can be estimated by measuring the average delay for a call to the web service,
- λ_d: arrival intensity of the default clients. This parameter can be estimated by the average arrivals per unit of time of default clients,
- λ_p: arrival intensity of premium clients,
- t: threshold on waiting time for premium clients,
- n: fraction of premium clients that should be serviced with a waiting time smaller than t.

The required outputs are the forwarding probabilities of the broker:

- p_i: the forwarding probability to web server i for a default client;
- p_{drop}: the probability of dropping a request from a default client. $\sum_i p_i + p_{drop} = 1$;
- q_i: the forwarding probability to web server i for a premium client. No premium client requests will be dropped, since the number of premium clients and the limit on premium client requests per second will be known from the SLAs; the servers should be dimensioned to take at least these limits into account. $\sum_i q_i = 1$

The algorithm is subject to the following SLA constraints:

- Ensuring no server is overloaded:

$$\lambda_i < \mu_i \tag{1}$$

- Ensuring the n-percentile, i.e. the probability of the waiting time for a premium client being smaller than the threshold t should be greater than n (with W_i the cumulative distribution function for the waiting time on server i):

$$\sum_i (q_i W_i(t)) \geqslant n \tag{2}$$

- Broker forwarding probabilities:

$$0 \leqslant p_i \leqslant 1, \quad 0 \leqslant q_i \leqslant 1, \quad 0 \leqslant p_{drop} \leqslant 1. \tag{3}$$

3.2. Modeling the SALSA objective

In order to model the different user profiles, two kinds of requests are considered. Premium clients require a SLA guaranteeing that the total waiting time for a request is less than a certain threshold, for a certain fraction of the requests (e.g. 95%). Premium requests should never be dropped. Default clients on the other hand do not require statistical guarantees and are served on a best effort basis. In order to ensure that premium requests are served within the threshold waiting time, default requests may be dropped. As a consequence, a trade-off needs to be made between

Please cite this article in press as: Boone, B., et al. SALSA: QoS-aware load balancing for autonomous service brokering. J. Syst. Software (2009), doi:10.1016/j.jss.2009.09.033

dropping default client requests and exceeding the premium threshold for more than the allowable fraction.

As a result, the objective of the SALSA broker algorithm is to minimize the average waiting time for all clients as well as the fraction of dropped requests, while upholding the contract for premium clients by guaranteeing the n-percentile, and ensuring no server is overloaded.

Since every server is modeled as a M/M/1 queueing system, Fig. 2 presents the effective arrival intensity for server i, assuming an arrival intensity λ:

$$\lambda_i = p_i\lambda_d + q_i\lambda_p \tag{4}$$

The following formulae for the average waiting time, and the n-percentile per server i can be easily derived through application of standard queueing theory:

- The cumulative distribution function for the waiting time on server i:

$$W_i(x) = 1 - e^{-(\mu_i-\lambda_i)x} \tag{5}$$

- The average waiting time (including the service time):

$$\bar{w}_i = 1/(\mu_i - \lambda_i) \tag{6}$$

- n-percentile waiting time:

$$w_i^n = \ln(1-n)/(\lambda_i - \mu_i) \tag{7}$$

The average waiting time for the total system, neglecting the delay in the broker itself, can then be found from:

$$\bar{w} = \sum_i (\lambda_i * \bar{w}_i) \Big/ \sum_i \lambda_i \tag{8}$$

The minimum for the SALSA objective can either be a local minimum inside the region of the solution space defined by the constraints or it can be found on the edge of the solution space. Both the expression for the average waiting time (Eq. (6)) and the constraint on the n-percentile (Eq. (2)) are non-linear, making theoretical treatment of the optimum difficult. If a local minimum is found in the inner region of the solution space, this minimum is guaranteed to be the minimum of the system. The derivatives of the average waiting time are, with p_1 substituted with $1 - p_{drop} - \sum_{i=2}^{k} p_i$:

$$\frac{\partial \bar{w}}{\partial p_i} = \frac{-\lambda_d\mu_1}{(\mu_1-\lambda_1)^2} + \frac{\lambda_d\mu_i}{(\mu_i-\lambda_i)^2} \tag{9}$$

$$\frac{\partial \bar{w}}{\partial q_i} = \frac{-\lambda_p\mu_1}{(\mu_1-\lambda_1)^2} + \frac{\lambda_p\mu_i}{(\mu_i-\lambda_i)^2} \tag{10}$$

A local extremum is found when:

$$\frac{\mu_i - \lambda_i}{\sqrt{\mu_i}} = \frac{\mu_1 - \lambda_i}{\sqrt{\mu_1}} \tag{11}$$

By using these equations, the SALSA objective can be tested for efficiency in the case the minimum is found in the inner region of the solution space. However, it is possible that the actual global minimum is on the bounds of the solution space; this should be checked using other means.

4. SALSA: simulated annealing based load spreading algorithm

This section discusses the SALSA algorithm, implementing the above defined objective. The strategy of the broker is to use forwarding probabilities in such a way that the average waiting time for each client is minimized, while at the same time ensuring that the n-percentile waiting time for premium clients is below the given threshold t, and avoiding dropped calls for default clients. In order to explore the solution space and find an optimum solution for the SALSA objective, Simulated Annealing is used.

4.1. Basic algorithm

Simulated annealing (SA) (Salamon, 2002; Kirkpatrick et al., 1983) is a generic probabilistic meta-algorithm for locating a good approximation to the global optimum of a given function in a large search space. Analogously to annealing in metallurgy, each step within the SA algorithm updates the current state to a random nearby state. During the SA algorithm a temperature parameter is gradually decreased and the next random state is chosen with a probability depending on the difference between the corresponding optimization function values, and the temperature parameter. The optimization function value of a state is analogous to the internal energy of a material in a certain state. The optimization function is therefore called the Internal Energy Function. The current state changes almost randomly when the temperature parameter is high (high temperature), but increasingly stabilizes as the temperature parameter goes to zero. The goal is to bring the system, from an arbitrary initial state, to a state with the minimum possible energy.

To evaluate a given set of forwarding probabilities, an Internal Energy Function is used to give a score, which is to be minimized. For each server, the actual arrival intensity is calculated, based on the forwarding probabilities. From the arrival intensity and processing intensity for the server, the average waiting time can be calculated using Eq. (6).

If the processing intensity is not larger then the arrival intensity ($\mu_i \leqslant \lambda_i$), the server will of course not be able to handle the load. Since this is unacceptable, the score is increased by a large constant (10^6), and additionally increased by the same large constant multiplied with a percentage of how severely the web service is overloaded. The latter helps the Simulated Annealing algorithm by differentiating between several undesirable solutions based on the quality of the resulting solution. If the server is not overloaded, the score is increased with the average waiting time ($\bar{w}$) divided by the threshold t, proportionally to the fraction of arrivals to this server, in order to minimize the average waiting time.

If less than a fraction n of the premium requests are serviced with a waiting time smaller than t, i.e. $\sum_i (q_i W_i(t)) < n$, a second component is added to the score, consisting of the fraction of requests which are not serviced in time multiplied with a constant *penaltyThreshold*. This accounts for the constraint in Eq. (2). Finally, a penalty is added to the score, proportional with the percentage of dropped calls.

$$score = \sum_i serverscore_i + thresholdscore + penaltyDrop \times p_{drop} \tag{12}$$

$$serverscore_i = \begin{cases} 10^6 \times \left(1 + \frac{(\lambda_i - \mu_i)}{\mu_i}\right) & \mu_i \leqslant \lambda_i \\ \frac{\lambda_i}{(1-p_{drop})\lambda_d + \lambda_p} \frac{\bar{w}_i}{t} & \text{otherwise} \end{cases} \tag{13}$$

$$thresholdscore = \begin{cases} penaltyThreshold \times \left(n - \sum_i frac_i\right) & \sum_i frac_i < n \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

$$frac_i = \begin{cases} 0 & \mu_i \leqslant \lambda_i \\ q_i W_i(t) & \text{otherwise} \end{cases} \tag{15}$$

The algorithm starts with random values for all p_i and q_i. From there, neighbor states are selected by choosing two random indices from either the p- or the q-array. The probability indexed by the

Please cite this article in press as: Boone, B., et al. SALSA: QoS-aware load balancing for autonomous service brokering. J. Syst. Software (2009), doi:10.1016/j.jss.2009.09.033

first one is increased with a given step size, and the probability indexed by the latter one is decreased with it. The step size is linearly dependent of the temperature, and thus decreases exponentially with the iteration number.

Algorithm 1. Random step function

success = *false*
while *success* = *false* **do**
  stepSize = $0.1 \times T/T_{start}$
  randomly choose $r = p$ or q
  randomly choose indexes i and j $(i \neq j)$
  if $r_i \geqslant stepSize$ **then**
    $r_i -=$ stepSize
    $r_j +=$ stepSize
    $success = true$
end

4.2. Tuning the algorithm

4.2.1. Number of iterations

A simulated annealing algorithm can run endlessly. However we assume that the algorithm converges after a number of iterations and stop after a fixed amount of iterations. This amount, and the convergence of the algorithm, is investigated in Section 5.3.4.

4.2.2. Penalty factors

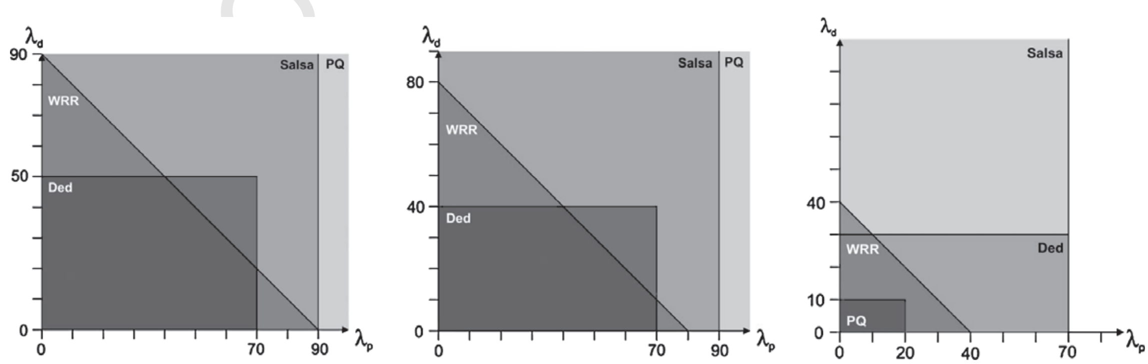
The ability of the algorithm to successfully identify constraint-meeting solutions depends on the penalty factors. Two configurable parameters are present in the algorithm: *penaltyThreshold* and *penaltyDrop*. *penaltyThreshold* controls the penalty associated with exceeding the n-percentile threshold for premium clients. *penaltyDrop* controls the penalty associated with dropping calls from default clients. Since dropping more calls will leave more headroom for meeting the threshold requirements, and relaxing the threshold requirements will enable the algorithm to find solutions with less dropped calls, these penalties provide a trade-off between dropping default clients and exceeding thresholds. Choosing an appropriate value for these penalties will depend on the application.

5. Evaluation results

The simulated annealing load balancing algorithm (see Section 4) is implemented to evaluate its correctness and its performance. In the first evaluation, we compare the applicability of the SALSA algorithm with several other load balancing algorithms for a number of server setups. The second evaluation is set in a highly controlled simulation environment, where especially the correctness of the mechanism is evaluated. The last evaluation is an experimental evaluation which uses several generated request patterns to stress-test a web service broker that can use a variety of load balancing algorithms. This experiment is especially set up to evaluate the differences between the SALSA 95%-priority-algorithm and weighted round-robin, and whether the algorithms can fulfill the goals set in the Introduction.

5.1. Applicability evaluation of SALSA

In the first set of evaluations we analytically calculate the response times given a particular load balancing algorithm and a particular server setup. In this evaluation, we require 95% of the response times of premium requests to be lower than 100 ms. The throughputs of both the premium and the default requests are discretely varied between 0 and 150 requests/s. By interpolating these calculated results, we can determine the area where the 95-percentile of the response times of the premium requests is lower than the threshold – we call this the applicability of that particular load balancing algorithms for that particular server setup. These calculations are done for three different server setups: (i) two very fast servers (10 ms and 20 ms); (ii) two distinct servers, but with their response times well under the threshold (9 ms and 28 ms); and (iii) three very different servers with one server close to the threshold (10 ms, 50 ms and 90 ms). The results are shown in Fig. 3 for the following load balancing algorithms: SALSA, weighted round-robin (WRR), dedicated server (Ded) and priority queue (PQ). We notice that the applicability of our SALSA load balancing algorithm is in most cases better or as good as the applicability of the other algorithms. Only the priority queue algorithm can outperform SALSA. However, in the trivial case with at least one extremely slow server (cf. server setup (iii)), the priority queue algorithm is no match for our SALSA algorithm – not even for the other evaluated load balancing algorithms. The applicability of the dedicated server solution and weighted round-robin is much stricter than that of the SALSA algorithm. As can be see from Fig. 3, weighted round-robin provides on average good results with a wide variation in throughputs (for premium and default requests). That is why, in the following sections, we will describe an in-depth comparison of the performance of the SALSA algorithm to the weighted round-robin load balancing algorithm using simulation and testbed evaluation.



(a) Two very fast servers (10 ms and 20 ms)

(b) Two distinct servers, but with their response times well under the threshold (9 ms and 28 ms)

(c) Three very different servers with one server close to the threshold (10 ms, 50 ms and 90 ms)

Fig. 3. Applicability of the SALSA algorithm compared to weighted round robin (WRR), dedicated server (Ded) and priority queue (PQ) load balancing for three cases.

Please cite this article in press as: Boone, B., et al. SALSA: QoS-aware load balancing for autonomous service brokering. J. Syst. Software (2009), doi:10.1016/j.jss.2009.09.033

5.2. Performance evaluation of SALSA

Within the simulation evaluation, the theoretical performance of SALSA is evaluated and the optimality of the solutions that the SALSA algorithm returns, is validated using the optimality criterion of Section 3.2. A Poisson arrival process is assumed for both default and premium clients, for which both λ_d and λ_p, respectively are known.

Within the simulation, the SALSA algorithm is run for given λ_d and λ_p, and based on the resulting p_i, q_i and λ_i, multiple performance attributes are calculated.

5.2.1. Inputs

In order to easily validate the results of the testbed experiment in Section 5.3 to the simulation results, the average service times in the simulation are chosen accordingly to the average service times of the web services used in the experimental evaluation. The experiment consists of two servers offering average service times of 9 ms and 28 ms, corresponding to $\mu_1 = 111.11$ req/s and $\mu_2 = 35.71$ req/s, respectively. As a consequence, the maximum throughput this system can handle is 146.82 req/s. As a result, λ_d and λ_p are varied in the simulation between 0 and this maximum. The threshold value t is set to 100 ms, and the percentile n set to 0.95.

5.2.2. Optimality of the returned SALSA results

First, a test was run to determine the optimality of the results returned by the SALSA algorithm. For this, the optimality criterion obtained in Section 3.2 was used. Fig. 4 shows the value of $|\frac{\mu_2-\lambda_2}{\sqrt{\mu_2}} - \frac{\mu_1-\lambda_i}{\sqrt{\mu_1}}|$, which should be zero if an optimum is found in the inner region of the problem's solution space, as a local extremum will satisfy $\frac{\mu_i-\lambda_i}{\sqrt{\mu_i}} = \frac{\mu_1-\lambda_i}{\sqrt{\mu_1}}$.

For small values of λ_d and λ_p, i.e. $\lambda_d + \lambda_p < 50$, the optimality measure differs from zero. Inspection of the returned p_i and q_i for these values shows that $p_i = 0$ or $q_i = 0$ for one of the servers i. An exhaustive search was conducted in this area, and no local extrema were found inside the solution space. This means that the optimum has to be found on the edge of the solution space (were the optimality criterion derived in Section 3.2 is different from zero). The SALSA algorithm found the optimum on the edge of the solution space and forwarded all requests to the same server.

For large values of λ_d and λ_p, i.e. $\lambda_d + \lambda_p > 90$, the optimality measure again differs from zero; here the algorithm finds an optimum on the boundaries that model the constraints on server load or exceeding thresholds. There are no solutions that fit the optimality criterion and that also fall within these constraints.

In between these regions, the optimality measure is close to zero, confirming the optimality of the results from the SALSA algorithm.

In order to further evaluate the optimality of the returned SALSA results, the same simulation was run with the penalties for dropping clients and exceeding priority thresholds set to zero. This effectively eliminates the corresponding boundaries on the solution space. The results are shown in Fig. 5. In this test, the optimality measure stays also close to zero for large values of λ_d and λ_p. For the area with small values of λ_d and λ_p, only an exhaustive search could confirm the optimality of the results.

From these results, we can conclude that our Simulated Annealing based algorithm is indeed able to find optimal results.

5.2.3. Performance of SALSA compared to weighted round-robin (WRR)

Another simulation was done to compare the performance of the SALSA algorithm with weighted round-robin. The weighted round-robin algorithm was run for the same given λ_p, λ_d. The arrival intensity for server i is calculated:

$$\lambda_i = (\lambda_p + \lambda_d) * \frac{\mu_i}{\sum_i \mu_i}$$

For this test, the fraction of clients serviced with a service time below the threshold t was calculated. Figs. 6 and 7 show the results for the SALSA algorithm and weighted round-robin respectively. In both graphs, a contour line is plotted for the n-percentile value of 0.95. As can be seen on Fig. 6, the SALSA algorithm can guarantee the 95-percentile for $\lambda_p < 90$, irrespective of the value for λ_d. Using the weighted round-robin algorithm (Fig. 7), the system fails to meet the 95-percentile for much smaller values of λ_p and λ_d. Both λ_p and λ_d have an influence on this, so that a high amount of default clients can deny QoS to the premium clients. Furthermore, if no special precautions are taken, the system gets overloaded when $\lambda_d + \lambda_p \geqslant 147.82$.

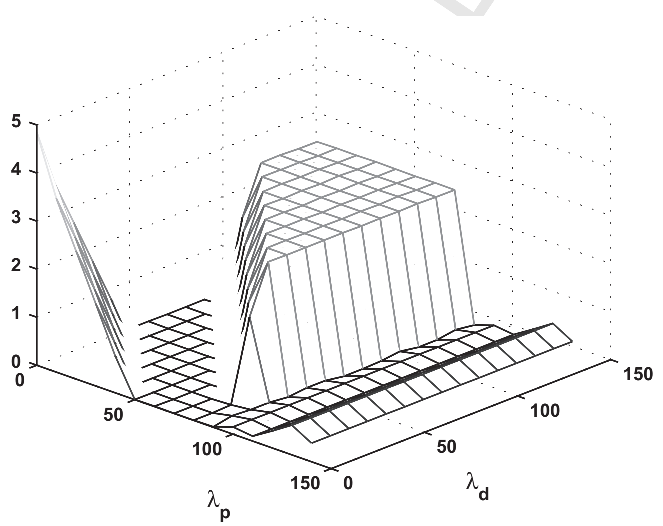


Fig. 4. Optimality measure for the SALSA algorithm. For small values of λ_d and $\lambda_p (\lambda_d + \lambda_p < 50)$, and large values of λ_d and $\lambda_p (\lambda_d + \lambda_p > 90)$, the optimality measure is different from zero; here the SALSA algorithm found a better solution on one of the boundaries of the solution space. For the values in between, the optimality measure is close to zero, which proves the optimality of the solution in this region.

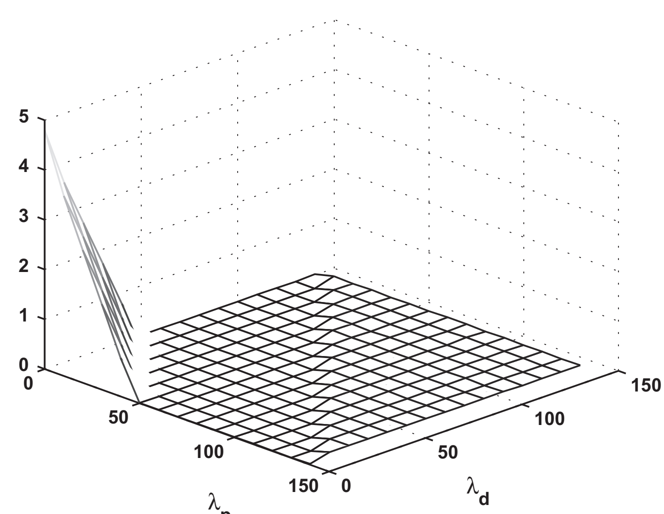


Fig. 5. Optimality measure for the SALSA algorithm, where the penalties for dropping clients and exceeding priority thresholds are set to 0. Here, the optimality measure is also close to zero for large values of λ_d and λ_p.

Please cite this article in press as: Boone, B., et al. SALSA: QoS-aware load balancing for autonomous service brokering. J. Syst. Software (2009), doi:10.1016/j.jss.2009.09.033

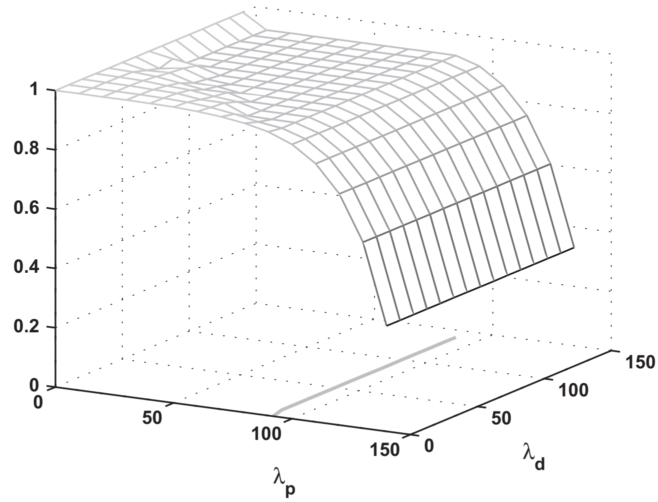


Fig. 6. Fraction of premium clients whose service time is below t, using the SALSA algorithm. The 95-percentile can be guaranteed for $\lambda_p < 90$, irrespective of the value for λ_d.

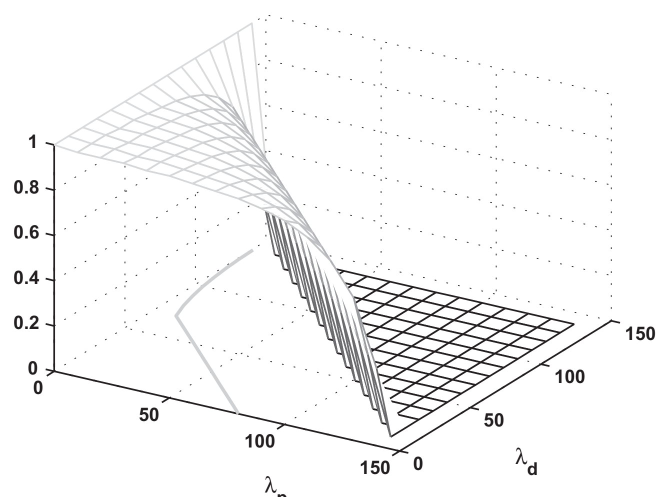


Fig. 7. Fraction of premium clients whose service time is below t, using the WRR algorithm. The system fails to meet the 95-percentile for smaller values of λ_p and λ_d than with SALSA.

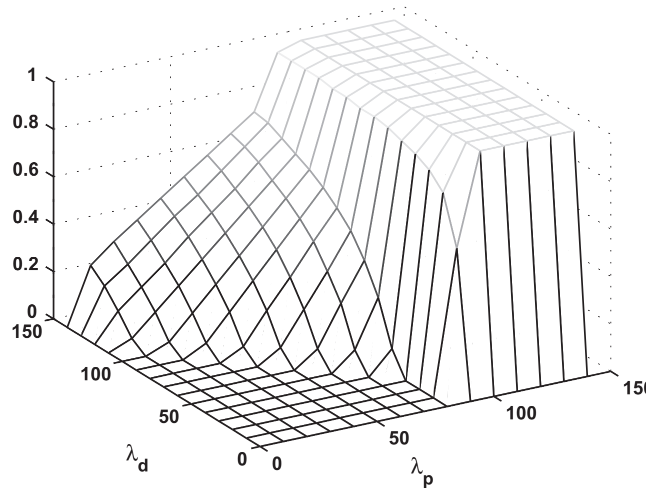


Fig. 8. Fraction of dropped calls with the SALSA algorithm.

5.2.4. Fraction of dropped default requests

Fig. 8 shows the fraction of dropped calls for the SALSA algorithm for varying λ_p and λ_d.

5.3. Testbed evaluation of SALSA

For this experiment, a prototype web service broker has been implemented. The testbed configuration and results are discussed in this section.

5.3.1. Testbed configuration

The test setup for the experimental evaluation, shown in Fig. 9, consists of three important components: a load generator, two web servers and the web service broker.

The load generator simulates real user behavior as Poisson processes realizing different request patterns for the two classes of users (premium and default customers).

The web servers both expose one Axis2 (Axis2/Java, xxxx) web service, with an average service time of respectively 9 ms and 28 ms.

The web services exposed by the web servers are purely computational services. As a consequence, their execution time is directly proportional to the amount of service requests (see Fig. 10). This behavior is in agreement with the modeling approach taken in Section 3.1. For not purely computational web services, for

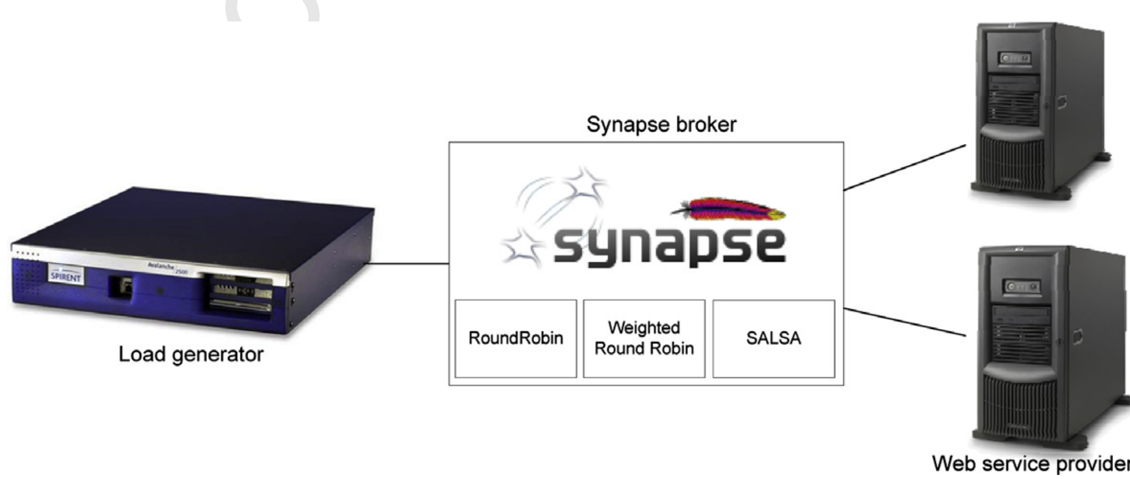


Fig. 9. Test setup for evaluation.

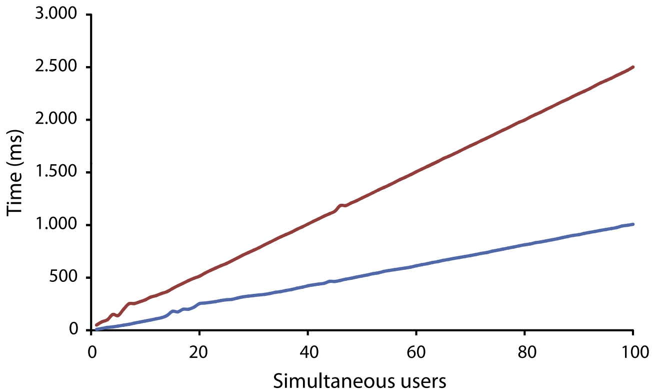


Fig. 10. Stress-testing both computational web services.

example services doing I/O as well, the assumption of an M/M/1 queueing system will be an overestimation, resulting in slightly over-dimensioning the load using SALSA. The web service broker is implemented using Apache Synapse (Synapse, xxxx), a lightweight and high performance Enterprise Service Bus (ESB). The Synapse engine comes with a set of transports, mediators and standard brokering capabilities, such as round-robin load balancing and fail-over. Some additional mediators are implemented to support different load balancing algorithms such as weighted round-robin and QoS monitoring. In order for the 95%-priority load balancing algorithm to work properly the optimization component used in the simulations of the previous subsection is also incorporated in the web services broker. This component will provide the optimized load distribution to the broker for use in its load balancing strategy. In order to minimize the possible system impact of both the ESB core functionality and the optimization component, running the presented SALSA algorithm, they execute at the top level in separate threads. To further avoid potential resource constraints, the web service broker is deployed on an extremely powerful Linux server with a multi-core AMD Opteron™ processor designed for optimum multi-threaded application performance.

5.3.2. Confidence intervals

The throughput of premium and default requests is needed as an input to the SALSA algorithm. Contrary to simulation, in real world scenario's these throughputs are unknown. Since Poisson arrival processes can have fluctuating arrival intensities, confidence intervals are used to estimate these arrival intensities and indicate the reliability of the estimates (Clopper and Pearson, 1934). The confidence level sets the boundaries of a confidence interval. In order to guarantee a 95th percentile to premium users, the confidence level for the arrival intensity needs to be 97.5% as well as the optimizing threshold within SALSA. Combining both estimates, a 95th percentile can be guaranteed to premium users. The 97.5% confidence level, with 0% area in the lower tail and 2.5% area in the upper tail, can be constructed using the χ^2-distribution with risk level $\alpha = 0.025$ (i.e. $97.5 = 100 * (1 - \alpha)$). Based on a 97.5% confidence level, a sample rate of 100 incoming messages is at least needed. Whenever the algorithm needs an estimate of the current throughput, the throughput over the last 100 arrivals is calculated and used as input for the SALSA algorithm.

5.3.3. Input request patterns

Since a commonly used model for random, mutually independent message arrivals is the Poisson process, the first input request pattern are two Poisson processes, one for the default requests and one for the premium requests, with variable arrival rate λ_d and λ_p, respectively.

Using Poisson arrival processes, extreme conditions such as a particular time period exhibiting an abnormally large number of events (Poisson burst), or contrary no events at all, are possible. Although within Poisson processes bursts can appear, a second input request pattern is used to explicitly evaluate the capabilities of the load balancing algorithm to handle request bursts on top of the Poisson bursts. Within a burst both λ_d and λ_p are increased at once. The period of the burst varies in the configured request pattern.

5.3.4. Number of iterations

In order to know after how many iterations on average the algorithm will show no improvements on the forwarding probabilities, a simulation has been conducted that uses different setups with an increasing number of iterations. Four setups were chosen, using 2–5 servers. Fig. 11 shows the scores for the solutions obtained for the different setups. From the results it is shown that after 100,000 iterations, the algorithm converges, and shows no more improvements on the resulting QoS. In our experimental setup, this takes about 2 s.

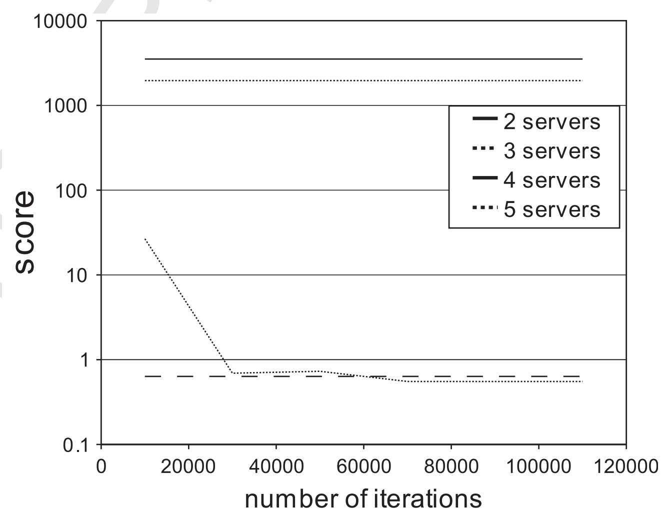


Fig. 11. Scores of different setups, in logarithmic scale. After 100,000 iterations, the algorithm shows no more improvements.

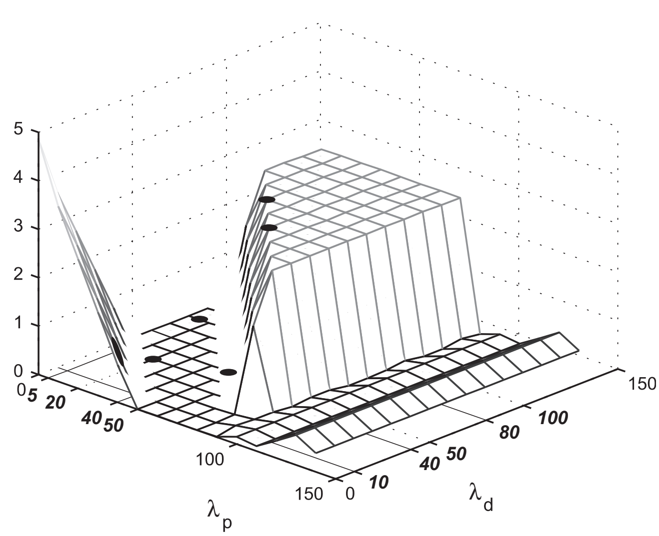


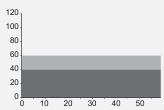
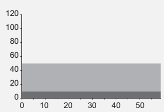
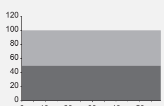
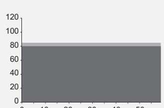
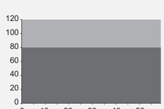
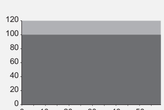
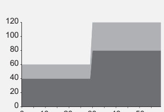
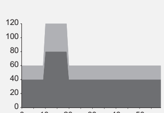
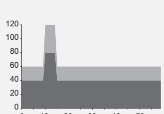
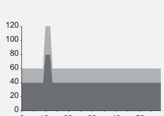
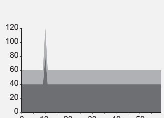
Fig. 12. Optimality measure for the SALSA algorithm showing the chosen samples for comparing SALSA and WRR in Table 1.

Please cite this article in press as: Boone, B., et al. SALSA: QoS-aware load balancing for autonomous service brokering. J. Syst. Software (2009), doi:10.1016/j.jss.2009.09.033

5.3.5. Penalty factors

To guide the simulated algorithm into the direction of constraint-meeting solutions, penalty factors are applied to solutions that does not meet the constraints. A proper setting of these penalty factors is important, because a large penalty factor sorts out non-constraint-meeting solutions very quickly while a low penalty can result in non-constraint-meeting solutions to be the result at the end. The penalties provide a trade-off between dropping default clients and exceeding thresholds. In the algorithm, the penalties are applied to the fraction of dropped clients, and the fraction of premium clients who exceed the threshold waiting time, respectively. For this experiment, exceeding the threshold is penalized 10 times more than dropping default clients. Since the penalties have to be considerably higher than the expected penalties for

Table 1
Comparing SALSA to WRR. Notice that in several tests the 95th percentile is lower for WRR. But the main goal, at least 95% of the requests needs to be served within the threshold, is always met by the SALSA algorithm in contrast to WRR.

Input pattern	λ_d	λ_p	Algorithm	95% (ms)	Crossing threshold (%)
	40	20	SALSA WRR	53 51	1.66 0.71
	10	40	SALSA WRR	54 53	2.28 0.59
	50	50	SALSA WRR	59 201	0.47 28.70
	80	5	SALSA WRR	63.9 54.7	2.47 0.75
	80	40	SALSA WRR	93.3 201	4.76 26.81
	100	20	SALSA WRR	57 77	1.314 3.31
	40–80	20–40	SALSA WRR	67.8 152	2.16 17.15
	40–80	20–40	SALSA WRR	99 84	4.92 4.76
	40–80	20–40	SALSA WRR	62 71.9	1.89 2.89
	40–80	20–40	SALSA WRR	76 52	2.71 0.20
	40–80	20–40	SALSA WRR	94.1 76.7	4.94 2.92

Please cite this article in press as: Boone, B., et al. SALSA: QoS-aware load balancing for autonomous service brokering. J. Syst. Software (2009), doi:10.1016/j.jss.2009.09.033

the waiting time, i.e. $\frac{\lambda_i}{(1-p_{drop})\lambda_d+\lambda_p}\frac{\bar{w}_i}{t}$, and considerably lower than the penalty for overloading the server (10^6) in order to never chose a solution with overloaded servers above a bad solution which does not overload the servers, the penalties in this experiment were chosen:

$$penaltyDrop = 1000$$
$$penaltyThreshold = 10,000$$

5.3.6. Comparison to weighted round-robin

In order to compare the performance of our simulated annealing algorithm, the weighted round-robin algorithm (WRR) is run in the same experiment setup. As can be seen in Fig. 12, within these tests, some of the λ_d and λ_p were chosen such that the optimum is found in the inner region of the solution space, while others where chosen such that the optimum is on the boundaries. In both cases however, SALSA is able to guarantee the 95th percentile.

The detailed results are shown in Table 1.

As can be seen in this table the weighted round-robin slightly outperforms the SALSA algorithm in underloaded circumstances. This is normal since all requests on the web service broker are of the same kind. As a result, weighted round-robin performs very well, while the SALSA load balancing algorithm requires more processing resulting in lower responsiveness. Since SALSA allows for autonomous brokering, the real arrival intensities are estimated using a confidence interval, resulting in the SALSA algorithm slightly over-dimensioning and being outperformed by WRR in underloaded circumstances. If however many prior requests need to be handled and the platform gets overloaded, the SALSA algorithm is able to guarantee the 95% to the prior requests while weighted round-robin crosses the threshold for more than 5% of the requests. Both SALSA and WRR can handle bursts. However, for long-term bursts, SALSA notices the higher arrival rates and immediately adjusts the load balancing in order to guarantee the QoS requirements, contrary to WRR. As a result, we can conclude that SALSA is able to dynamically adapt its load balancing strategy to handle dynamic request patterns without a priori over-dimensioning the web servers' resources in order to guarantee the SLAs to premium customers.

6. Conclusion and future work

By using the SALSA algorithm, requiring slightly more processing than weighted round-robin, brokers can guarantee a n-th percentile response time to their premium users, while providing best effort to the default customers. As service-oriented architectures have largely distributed topologies, SOA broker architectures can benefit from our SALSA algorithm as the service providers can be QoS unaware, released from mediating SLAs, and don't have to be a priori over-dimensioned. SALSA provides QoS-aware load balancing for autonomous service brokering since the SLAs are only mediated between the customers and the QoS-aware broker. To this end, the SALSA algorithm divides the load taking into account a real time view of the requests by measuring the arrival rates at that moment. If needed, requests from the default users will be dropped to reduce the web servers' load in order to guarantee the SLA to premium customers. By using Business Activity Monitoring, providing real time information about the status of service processes and transactions, the decision-making process within SALSA can be improved by using the derived intelligence to analyze and improve the efficiency of the load balancing. Business Activity Monitoring provides brokers with the ability to instrument their services to monitor events, correlate these events with each other and to understand their impact on the Key Performance Indicators. We will continue the design of advanced load balancing algorithms, fulfilling QoS requirements and optimize the decision-making within the SALSA algorithm by using Business Activity Monitoring.

Acknowledgments

Part of this work is supported by the Research Foundation Flanders (Fonds Wetenschappelijk Onderzoek Vlaanderen) in the context of the DyBroWS project on "Intelligent dynamic brokering of Web services".

Sofie Van Hoecke and Gregory Van Seghbroeck would like to thank the IWT (Institute for the Promotion of Innovation through Science and Technology in Flanders), and Bas Boone would like to thank the BOF (Bijzonder Onderzoeksfonds) of Ghent University, for financial support through their Ph.D. grant.

References

Ardagna, D., Ghezzi, C., Mirandola, R., 2008. Model driven QoS analyses of composed web services. In: LNCS Proceedings of the 1st European Conference on Towards a Service-Based Internet, Madrid, Spain.
Apache Axis2/Java. <http://ws.apache.org/axis2/>.
Bryhni, H., Klovning, E., Kure, O., 2000. A comparison of load balancing techniques for scalable web servers. IEEE Network, 58–64.
Cardellini, V., Colajanni, M., Yu, P.S., 1999. Load balancing on web-server systems. IEEE Internet Computing 3 (3), 28–39.
Christodoulopoulos, K., Varvarigos, M., Develder, C., De Leenheer, M., Dhoedt, B., 2007. Job demand models for optical grid research. In: Proceedings of the 11th Conference on Optical Network Design and Modelling (ONDM), Athens, Greece.
Clopper, C.J., Pearson, E.S., 1934. The use of confidence or fiducial limits illustrated in the case of the binomial. Biometrika 26, 404–413.
Cortes, A., Ripoll, A., Senar, M.A., Luque, E., 1999. Performance comparison of dynamic load-balancing strategies for distributed computing. In: Proceedings of the 32nd Annual Hawaii International Conference on System Sciences (HICSS-32).
Di Stefano, A., Lo Bello, L., Tramontana, E., 1999. Factors affecting the design of load balancing algorithms in distributed systems. Journal of Systems and Software 48 (2), 105–117.
Garofalakis, J., Panagis, Y., Sakkopoulos, E., Tsakalidis, A., 2006. Contemporary web service discovery mechanisms. Journal of Web Engineering 5 (3), 265–290.
Gross, D., Harris, C., 1998. Fundamentals of Queueing Theory, 3rd ed. Q4
Grosu, D., Chronopoulos, A.T., Leung, A.T., 2002. Load balancing in distributed systems: an approach using cooperative games. In: Proceedings of the Parallel and Distributed Processing Symposium.
Kanodia, V., Knightly, E.W., 2000. Multi-class latency-bounded web services. In: Proceedings of the IEEE/IFIP International Workshop on Quality of Service (IWQoS).
Kephart, J.O., Chess, D.M., 2003. The vision of autonomic computing. Computer 36 (1), 41–50.
Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983. Optimization by Simulated Annealing, Science, Number 4598 220 (4598), 671–680.
Levy, R., Nagarajarao, J., Pacici, G., Spreitzer, M., Tantawi, A., Youssef, A., 2003. Performance management for cluster based web services. In: IFIP/IEEE Eighth International Symposium on Integrated Network Management, vol. 246, pp. 247–261.
Liu, Y., Ngu, A.H.H., Zeng, L., 2004. QoS computation and policing in dynamic web service selection. In: Proceedings of the WWW'04.
Nahrstedt, K., Smith, J.M., 1995. The QoS broker. IEEE Multimedia Magazine 2 (1).
Nami, M.R., Bertels, K., 2007. A survey of autonomic computing systems. In: Proceedings of the Third International Conference on Autonomic and Autonomous Systems (ICAS07).
Roberts, J.W., 2001. Traffic theory and the internet. IEEE Communications Magazine 39 (1), 94–99.
Salamon, P., 2002. Facts, Conjectures And Improvements For Simulated Annealing. SIAM Monographs on Mathematical Modeling and Computation. Society for Industrial and Applied Mathematics, US.
Shirazi, B.A., Hurson, A.R., Kavi, K.M. (Eds.), 1995. Scheduling and Load-Balancing in Parallel and Distributed Systems. IEE CS Press.
Sterritta, R., Parasharb, M., Tianfieldc, H., Unlandd, R., 2005. A concise introduction to autonomic computing. Advanced Engineering Informatics Autonomic Computing 19 (3), 181–187.
Apache Synapse. <http://synapse.apache.org>.
Tao, Y., Lin, K.J., 2004. The design of QoS broker algorithms for QoS-capable web services. In: Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service.

Please cite this article in press as: Boone, B., et al. SALSA: QoS-aware load balancing for autonomous service brokering. J. Syst. Software (2009), doi:10.1016/j.jss.2009.09.033

Verheecke, B., 2007. Ph.D. Thesis, Dynamic Integration, Composition, Selection and Management of Web Services in Service-Oriented Applications, Department of Computer Science, System and Software Engineering Lab, Vrije Universiteit Brussel, 2007.

Zhang, Y., Harrison, P.O., 2007. Performance of a priority-weighted round robin mechanism for differentiated service networks. In: Proceedings of the 16th International Conference on Computer Communications and Networks (ICCCN '07), Honolulu.

Zhang, J., Hamalainen, T., Joutsensalo, J., Kaario, K., 2001. QoS-aware load balancing algorithm for globally distributed web systems. In: Proceedings of the info-tech and info-net (ICII01), Beijing.

Please cite this article in press as: Boone, B., et al. SALSA: QoS-aware load balancing for autonomous service brokering. J. Syst. Software (2009), doi:10.1016/j.jss.2009.09.033